4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

Over-Approximating Terms Reachable by Context-Sensitive Rewriting (full version)

Nirina Andrianarivelo and Pierre Réty
LIFO, Université d'Orléans

Rapport n^o **RR-2015-02**

Over-Approximating Terms Reachable by Context-Sensitive Rewriting

Nirina Andrianarivelo and Pierre Réty

LIFO - Université d'Orléans, B.P. 6759, 45067 Orléans cedex 2, France
{Nirina.Andrianarivelo, Pierre.Rety}@univ-orleans.fr

Abstract. For any left-linear context-sensitive term rewrite system and any regular language of ground terms I, we build a finite tree automaton that recognizes a superset of the descendants of I, i.e. of the terms reachable from I by context-sensitive rewriting.

1 Introduction

There is an increasing need for reliable methods to check security protocols and computer programs (see [3, 4] for a survey). Such verification problems can often be encoded with rewrite rules, and reduced to reachability problems [2]. Given a set of rewrite rules R, a set of ground terms I, and a set of undesirable ground terms BAD, it consists in computing the set (denoted $R^*(I)$) of ground terms that are reachable from I by R, and checking that $R^*(I) \cap BAD = \emptyset$.

Example 1. Let $R = \{f(x) \to f(f(x)),\ a \to b,\ c \to a\}$, $I = \{f(a)\}$, $BAD = \{c\}$. Then $R^*(I) = \{f^+(a), f^+(b)\}$, where f^+ denotes several occurrences of f (at least one). Here the elements of BAD are not reachable, i.e. $R^*(I) \cap BAD = \emptyset$.

Several methods have been proposed to compute $R^*(I)$ exactly, or to over-approximate it (see [10] for a survey). However, ordinary term rewriting is not always powerful enough. Indeed, the operational semantics of functional programs can be expressed using Context-Sensitive Term Rewrite Systems (CS-TRS), as described in [8, 15]. In this framework, a set of argument numbers $\mu(f)$ is associated to each function symbol f, which indicates the arguments of f allowed to be reduced by rewriting. For instance, consider Example 1 again and let $\mu(f) = \emptyset$. In this case, for each term of the form $t = f(\ldots)$, it is forbidden to rewrite the strict subterms of t. Thus, using context-sensitive rewriting, $R^*_\mu(I) = \{f^+(a)\}$. Now consider that the set of undesirable terms is $BAD' = \{f^+(b)\}$, then $R^*_\mu(I) \cap BAD' = \emptyset$ whereas $R^*(I) \cap BAD' \neq \emptyset$.

In this paper, we compute an over-approximation (say *App*) of $R^*_\mu(I)$, using a finite tree automaton (i.e. a regular language), assuming that R is left-linear and I is regular. Thus, if $App \cap BAD = \emptyset$, we are sure that $R^*_\mu(I) \cap BAD = \emptyset$. Our work is both an extension of [14], where $R^*_\mu(I)$ is computed in an exact way assuming stronger restrictions (R is linear and right-shallow[1]), and an extension

[1] I.e. in the rewrite rule right-hand-sides, every variable occurs at depth at most 1.

of the completion of tree automata [9], in order to take μ into account and to avoid computing descendants forbidden by μ (as much as possible).

Let us outline the main idea, using Example 1 again. Roughly[2]:

1. Our starting point is composed of the tree automaton $\mathcal{A} = (\Sigma, Q, Q_f, \Delta)$ s.t. $\Sigma = \{f, a, b, c\}$, $Q = \{q_a, q\}$ (states), $Q_f = \{q\}$ (final state), $\Delta = \{a \rightarrow q_a,\ f(q_a) \rightarrow q\}$ (transitions), which recognizes $I = \{f(a)\}$, and the rewrite system $R = \{f(x) \rightarrow f(f(x)),\ a \rightarrow b,\ c \rightarrow a\}$.
2. **Initialization.** We transform Δ by using the fact that $\mu(f) = \emptyset$. For this, we *mark* positions forbidden by μ, using a *a prime mark (priming)*. A prime means that no rewrite step should be applied at this position. No prime means that a rewrite step (if any) is allowed. Thus we replace the transition $f(q_a) \rightarrow q$ by $f(q'_a) \rightarrow q$, and add the transition $a \rightarrow q'_a$ so that the language recognized by the automaton is unchanged.
Now $\Delta = \{a \rightarrow q_a,\ a \rightarrow q'_a,\ f(q'_a) \rightarrow q\}$, and we create $Q' = \{q'_a\}$.
3. **Completion.** To get descendants, we compute (so-called) critical pairs between transitions $u \rightarrow s \in \Delta$ and rewrite rules of R, only if $s \in Q$. If $s \in Q'$, i.e. s has a prime, no critical pair is computed since no rewrite step is allowed at this position.

   Computing a critical pair between $a \rightarrow q_a$ and $a \rightarrow b$ generates the transition $b \rightarrow q_a$, which is added into Δ.
4. Computing a critical pair between $f(q'_a) \rightarrow q$ and $f(x) \rightarrow f(f(x))$ generates the transition $f(f(q'_a)) \rightarrow q$. However, $f(f(q'_a))$ is not shallow. Then we *normalize*[3] this transition into $f(q'_1) \rightarrow q, f(q'_a) \rightarrow q'_1$, which are added into Δ. The new state q'_1 has a prime since it occurs at a position forbidden by μ. Now the automaton also recognizes $f(f(a))$, and does not recognize $f(f(b))$, which is not a context-sensitive descendant.
5. There are some more critical pairs, which add some more transitions into Δ. Finally, the completion process stops with $\Delta =$

$$\{a \rightarrow q_a,\ a \rightarrow q'_a,\ f(q'_a) \rightarrow q,\ b \rightarrow q_a,\ f(q'_1) \rightarrow q, f(q'_a) \rightarrow q'_1,\ f(q'_1) \rightarrow q'_1\}$$

   Now the current automaton recognizes $\{f^+(a)\}$, i.e. $R^*_\mu(I)$, and does not recognize the elements of $\{f^+(b)\}$, which are not context-sensitive descendants.

The paper is organized as follows. The formal preliminary notions are given in Section 2. Our method for over-approximating context-sensitive descendants is detailed in Section 3, and full examples are given in Section 4. Then Section 5 speaks about related work, and some ideas for further work are discussed in Section 6. All proofs are in the appendix.

[2] Some unnecessary transitions of the current automaton are missing.
[3] It consists in *flattening* the left-hand-side of the transition by using intermediate states.

2 Preliminaries

Consider a *finite ranked alphabet* Σ and a set of variables Var. Each symbol $f \in \Sigma$ has a unique arity, denoted by $ar(f)$. The notions of *first-order term*, *position* and *substitution* are defined as usual. Given σ and σ' two substitutions, $\sigma \circ \sigma'$ denotes the substitution such that for any variable x, $\sigma \circ \sigma'(x) = \sigma(\sigma'(x))$. T_Σ denotes the set of ground terms (without variables) over Σ. For a term t, $Var(t)$ is the set of variables of t, $Pos(t)$ is the set of positions of t, and ϵ is the root position. For $p \in Pos(t)$, $t(p)$ is the symbol of $\Sigma \cup Var$ occurring at position p in t, and $t|_p$ is the subterm of t at position p. For $p, p' \in Pos(t)$, $p < p'$ means that p occurs in t strictly above p'. The term t is *linear* if each variable of t occurs only once in t. The term $t[t']_p$ is obtained from t by replacing the subterm at position p by t'.

A *rewrite rule* is an oriented pair of terms, written $l \to r$. We always assume that l is not a variable, and $Var(r) \subseteq Var(l)$. A *rewrite system* R is a finite set of rewrite rules. *lhs* stands for left-hand-side, *rhs* for right-hand-side. The *rewrite relation* $\to_R$ is defined as follows: $t \to_R t'$ if there exist a non-variable position $p \in Pos(t)$, a rule $l \to r \in R$, and a substitution θ s.t. $t|_p = \theta(l)$ and $t' = t[\theta(r)]_p$ (also denoted $t \to_R^p t'$). $\to_R^*$ denotes the reflexive-transitive closure of $\to_R$. t' is a *descendant* of t if $t \to_R^* t'$. If I is a set of ground terms, $R^*(I)$ denotes the set of descendants of elements of I. The rewrite rule $l \to r$ is *left (resp. right) linear* if l (resp. r) is linear. R is *left (resp. right) linear* if all its rewrite rules are left (resp. right) linear. R is *linear* if R is both left and right linear. $l \to r$ is *right-shallow* if r is *shallow*, i.e. every variable of r occurs at depth at most 1.

A *context-sensitive rewrite relation* is a sub-relation of the ordinary rewrite relation in which rewritable positions are indicated by specifying arguments of function symbols. A mapping $\mu : \Sigma \to P(\mathbb{N})$ is said to be a *replacement map* (or Σ-map) if $\mu(f) \subseteq \{1, ..., ar(f)\}$ for all $f \in \Sigma$. A *context-sensitive* term rewriting system (CS-TRS) is a pair $\mathcal{R} = (R, \mu)$ composed of a TRS and a replacement map. The set of μ-replacing positions[4] $Pos^\mu(t)$ ($\subseteq Pos(t)$) is recursively defined: $Pos^\mu(t) = \{\epsilon\}$ if t is a constant or a variable, otherwise $Pos^\mu(f(t_1, \dots, t_n)) = \{\epsilon\} \cup \{i.p \mid i \in \mu(f),\, p \in Pos^\mu(t_i)\}$. The rewrite relation induced by a CS-TRS $\mathcal{R}$ is defined: $t \hookrightarrow_{\mathcal{R}} t'$ if $t \to_R^p t'$ for some $p \in Pos^\mu(t)$. The set of descendants of I by context-rewriting according to the CS-TRS $\mathcal{R} = (R, \mu)$ is denoted $R_\mu^*(I)$.

A (bottom-up) finite tree *automaton* is a quadruple $\mathcal{A} = (\Sigma, Q, Q_f, \Delta)$ where Q is the set of states, $Q_f \subseteq Q$ is the set of final states, and Δ is a set of *transitions* of the form $t \to q$ where $t \in T_{\Sigma \cup Q}$ and $q \in Q$. A transition is *normalized* if it is of the form $f(q_1, \dots, q_n) \to q$ where $f \in \Sigma$ and $q_1, \dots, q_n, q \in Q$, or of the form $q_1 \to q$ (*empty transition*, also called *epsilon transition*). $\mathcal{A}$ is normalized if all transitions in Δ are normalized. Sets of *states* will be denoted by letters Q, S, D, and states by q, s, d.

The rewrite relation induced by Δ is denoted by $\to_\Delta$ or $\to_{\mathcal{A}}$. A ground term t is *recognized* by $\mathcal{A}$ into q if $t \to_\Delta^* q$. Let $L(\mathcal{A}, q) = \{t \in T_\Sigma \mid t \to_\Delta^* q\}$. The language recognized by $\mathcal{A}$ is $L(\mathcal{A}) = \cup_{q \in Q_f} L(\mathcal{A}, q)$. A set I of ground terms is

[4] Also called positions allowed by μ.

regular if there exists a finite automaton $\mathcal{A}$ s.t. $I = L(\mathcal{A})$. A *Q-substitution* σ is a substitution s.t. $\forall x \in Dom(\sigma)$, $\sigma(x) \in Q$.

3 Computing Context-Sensitive Descendants

3.1 Closure under Context-Sensitive Rewriting

The main idea is: given a context-sensitive rewrite system (R, μ), we consider a set of states Q to recognize subterms at positions allowed by μ (i.e. rewritable positions), and another set Q' for those forbidden by μ. To compute context-sensitive descendants, rewrite steps will be applied to (sub)-terms recognized into states of Q, and not on those recognized into states of Q'.

Definition 1. A *context-sensitive automaton* $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$ is composed of a tree automaton and a mapping rm' such that $Q \cap Q' = \emptyset$, $Q_f \subseteq Q$, and $rm' : Q' \to Q$ is an injective mapping. rm' stands for 'remove primes'.

We will often use $q, q_1, q_2, \ldots$ for elements of Q, and $q', q'_1, q'_2, \ldots$ for elements Q', and we will write $rm'(q') = q$ and $rm'(q'_i) = q_i$.

Definition 2. We extend rm' to terms, so that $rm' : T_{\Sigma \cup Q \cup Q'} \to T_{\Sigma \cup Q \cup Q'}$, by:

- $rm'(q) = q$ if $q \in Q$,
- and $rm'(f(t'_1, \ldots, t'_n)) = f(t_1, \ldots, t_n)$ such that $\forall i \ \left| \begin{array}{l} t_i = rm'(t'_i) \text{ if } i \in \mu(f) \\ t_i = t'_i \text{ otherwise} \end{array} \right.$

Note that rm' does not remove all primes. Actually, rm' removes primes (if any) from states occurring at positions allowed by μ, so that rewrite steps are computed. For example, if $\mu(f) = \{1\}$, then $rm'(f(q'_1, q'_2)) = f(q_1, q'_2)$.

For computing context-sensitive descendants, a context-sensitive automaton should satisfy a compatibility property (with μ).

Definition 3. Let $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$ be a context-sensitive automaton. $\mathcal{A}$ is μ-compatible if $\forall (t \to s) \in \Delta$, $(rm'(t) \to rm'(s)) \in \Delta$.

Example 2. let $Q = \{q_a, q_f\}$, $Q' = \{q'_a\}$, $Q_f = \{q_f\}$, $\Delta = \{a \to q'_a,\ f(q'_a) \to q_f\}$, and assume that $rm'(q'_a) = q_a$ and $\mu(f) = \{1\}$. This automaton is not μ-compatible because $a \to q_a$ and $f(q_a) \to q_f$ are missing in Δ.

Lemma 1. *If $\mathcal{A}$ is μ-compatible,*
$\forall t \in T_{\Sigma \cup Q \cup Q'}, \forall s \in Q \cup Q'$, $(t \to^*_\Delta s \implies rm'(t) \to^*_\Delta rm'(s))$

The notion of critical pair is at the heart of the technique. A critical pair is a way to detect a possible rewrite step issued from a term $t \in L(\mathcal{A}, q)$, by a rewrite rule $l \to r$. To check that this rewrite step is allowed by μ, we suppose that $q \in Q$, i.e. $q \notin Q'$. A convergent critical pair means that the rewrite step is already handled i.e. if $t \to_{l \to r} s$ then $s \in L(\mathcal{A}, q)$. Consequently, the language of a normalized automaton having only convergent critical pairs is closed under rewriting.

Definition 4. Let $l \to r$ be a rewrite rule and σ be a $(Q \cup Q')$-substitution such that $\sigma l \to_{\Delta}^{*} q$ and $q \in Q$. Then $(rm'(\sigma r), q)$ is called *critical-pair* (CP for short). The critical pair is said *convergent* if $rm'(\sigma r) \to_{\Delta}^{*} q$.

Example 3. Consider Example 2 again, and the rewrite rule $f(x) \to g(x)$ with $\mu(g) = \{1\}$. Then $(g(q_a), q_f)$ is a critical pair, which is not convergent. Note that $L(\mathcal{A})$ is not closed by context-sensitive rewriting since $f(a) \in L(\mathcal{A})$ whereas $g(a) \notin L(\mathcal{A})$.

The use of rm' in Definition 4 is crucial if a position forbidden by μ becomes allowed after a rewrite step. For instance, consider the rewrite system $\{h(x) \to i(x),\, c \to d\}$ with $\mu(h) = \emptyset$ and $\mu(i) = \{1\}$. Then $h(c) \to i(c) \to i(d)$ whereas $\neg(h(c) \to h(d))$. So, within $h(c)$, c should be recognized into a state of Q' (say q'_c), whereas within $i(c)$, c should be recognized into a state of Q (say q_c). The migration of q'_c into q_c is achieved thanks to rm'.

To get closure under context-sensitive rewriting, the automaton should be μ-compatible to take μ into account, and normalized. Indeed, if it is not normalized, we may have for example $h(\sigma l) \to_{\Delta}^{*} q$ whereas $\neg(\exists q_1 \in Q,\, \sigma l \to_{\Delta}^{*} q_1)$, i.e. there is no critical pair to take the rewrite step by $l \to r$ into account.

Theorem 1. *Let (R, μ) be a left-linear context-sensitive rewrite system, and $\mathcal{A}$ be a μ-compatible normalized automaton.*
If all critical pairs are convergent, then $L(\mathcal{A})$ is closed by context-sensitive rewriting, i.e. $(t \in L(\mathcal{A}) \wedge t \hookrightarrow_{(R,\mu)}^{*} t') \Longrightarrow t' \in L(\mathcal{A})$.

Example 4. Consider Example 2 again, and the rewrite rule $a \to b$. All critical pairs are convergent since there are no critical pairs. However $f(a) \in L(\mathcal{A})$ and $f(a) \hookrightarrow_{(R,\mu)} f(b)$, whereas $f(b) \notin L(\mathcal{A})$. This comes from the fact that $\mathcal{A}$ is not μ-compatible.
Now, if Δ is replaced by $\Delta' = \{a \to q'_a,\, a \to q_a,\, b \to q_a,\, f(q_a) \to q_f\}$, the automaton is μ-compatible. Considering the rewrite rule $a \to b$, there is one critical pair : (b, q_a), which is convergent. Thus $f(a) \in L(\mathcal{A})$, $f(a) \hookrightarrow_{(R,\mu)} f(b)$, and $f(b) \in L(\mathcal{A})$.

3.2 Normalization

Consider a non-convergent critical pair (t, q). If we add the transition $t \to q$ into Δ, the critical pair becomes convergent. Unfortunately, the transition $t \to q$ is not necessarily normalized.

Example 5. Consider $R = \{f(x) \to g(h(x)),\, a \to b\}$, $\mu(f) = \{1\}$, $\mu(g) = \emptyset$, $\mu(h) = \{1\}$, and an automaton defined by $Q = \{q_a, q_f\}$, $Q_f = \{q_f\}$, and $\Delta = \{a \to q_a,\, f(q_a) \to q_f\}$. Note that $L(\mathcal{A}) = \{f(a)\}$. From the transition $f(q_a) \to q_f$ and the rewrite rule $f(x) \to g(h(x))$, we get the critical pair $(g(h(q_a)), q_f)$. The corresponding transition $g(h(q_a)) \to q_f$ is not normalized.

To get closure under rewriting, all transitions should be normalized. We give an algorithm to transform a pair (t, s) into normalized transitions. Note that if t is a state, the algorithm will return empty transitions (which are normalized).

Input : a pair (t, s) s.t $t \in T_{\Sigma \cup Q \cup Q'}$ and $s \in Q \cup Q'$
Output : a set of normalized transitions

function $\mathsf{Norm}_{\mathcal{A}}(t, s)$

1. If the transition $t \to s$ is normalized, return $\{t \to s\} \cup \{rm'(t) \to rm'(s)\}$
2. else let $t = f(t_1, \dots, t_n)$, and $J = \{j \in \{1, \dots, n\} \mid t_j \notin Q \cup Q'\}$
   2.1. for each $i \in \{1, \dots, n\}$, let s_i be a state defined by
      2.1.1. if $t_i \in Q \cup Q'$ then $s_i = t_i$
      2.1.2. else
         i) if $s \in Q$ and $i \in \mu(f)$
         ii) then either choose $s_i \in Q$, or s_i is a new state and add s_i to Q
         iii) else either choose $s_i \in Q'$, or s_i (and q_i) are new states s.t. $rm'(s_i) = q_i$ and add s_i to Q' (and q_i to Q)
   2.2. return
   $\{f(s_1, \dots, s_n) \to s\} \cup \{rm'(f(s_1, \dots, s_n)) \to rm'(s))\} \cup \{\cup_{j \in J} \mathsf{Norm}_{\mathcal{A}}(t_j, s_j)\}$

In the previous algorithm, whenever a transition is generated, a transition obtained by applying rm' on both sides is also generated. This is for preserving the μ-compatibility of the automaton. On the other hand, the non-determinism of the algorithm (Items ii and iii) is "don't care", i.e. only one choice has to be achieved. For any choice, the normalization algorithm terminates. However, introducing new states may create new critical pairs, whose normalization may also create new states and new critical pairs, and the global completion process may not terminate. This is why choosing s_i among the existing states is sometimes necessary to make completion terminate. But it may lead to a strict over-approximation of the descendants.

Example 6. Consider Example 5 again with the critical pair $(g(h(q_a)), q_f)$. Recall that $\mu(g) = \emptyset$, $\mu(h) = \{1\}$.
Running $\mathsf{Norm}_{\mathcal{A}}(g(h(q_a)), q_f)$ goes through the case iii), and two new states q_1', q_1 are created s.t. $rm'(q_1') = q_1$. Moreover q_1' is added to Q' whereas q_1 is added to Q. Then $\mathsf{Norm}_{\mathcal{A}}(g(h(q_a)), q_f)$ returns (note that $rm'(g(q_1')) = g(q_1')$):

$$\{g(q_1') \to q_f\} \cup \{g(q_1') \to q_f\} \cup \mathsf{Norm}_{\mathcal{A}}(h(q_a), q_1')$$

Since the transition $h(q_a) \to q_1'$ is already normalized, $\mathsf{Norm}_{\mathcal{A}}(h(q_a), q_1')$ returns:

$$\{h(q_a) \to q_1'\} \cup \{h(q_a) \to q_1\}$$

Finally we get the set of transitions $\{g(q_1') \to q_f,\ h(q_a) \to q_1',\ h(q_a) \to q_1\}$.

Lemma 2. $t \to^*_{\mathsf{Norm}_{\mathcal{A}}(t,s)} s$*, i.e. the pair* (t, s) *is convergent.*

3.3 Initialization

As in [14], we first introduce non-final states and transitions to recognize the ground subterms of the rewrite rule right-hand-sides. For a term t, let $PosG(t) =$

$\{p \in Pos(t) \mid p \neq \epsilon \wedge Var(t|_p) = \emptyset\}$. Let $PosGout(t)$ be the outermost elements of $PosG(t)$, i.e. $PosGout(t) = \{p \in PosG(t) \mid \neg(\exists p' \in PosG(t),\, p' < p)\}$.

Given (R, μ), we introduce the set of states $Q_R = \{q_{r,p} \mid l \to r \in R \wedge p \in PosG(r)\}$ and $Q'_R = \{q'_{r,p} \mid l \to r \in R \wedge p \in PosG(r)\}$ s.t. $rm'(q'_{r,p}) = q_{r,p}$, and the transitions $\Delta_R = \cup_{l \to r \in R} \cup_{p \in PosG(r)} \{r(p)(q'_{r,p.1}, \dots, q'_{r,p.n}) \to q'_{r,p},$ $rm'(r(p)(q'_{r,p.1}, \dots, q'_{r,p.n})) \to q_{r,p}\}$. Note that the transitions with rm' are for ensuring μ-compatibility.

From a normalized automaton $\mathcal{A}_0 = (\Sigma, Q_0, Q_f, \Delta_0)$ and a left-linear context-sensitive rewrite system (R, μ), a $\mu-$compatible normalized context-sensitive automaton $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$ that recognizes the same language as $\mathcal{A}_0$, is built as follows:

function $\mathsf{Init}_{(R,\mu)}(\mathcal{A}_0)$

1. for each $q \in Q_0$, a new state q' (also denoted $add'(q)$) is created, and let $rm'(q') = q$
2. extend add' to trees of $T_{\Sigma \cup Q_0}$: $add'(f(t_1, \dots, t_n)) = f(add'(t_1), \dots, add'(t_n))$
3. let $Q = Q_0 \cup Q_R$ and $Q' = \{add'(q) \mid q \in Q_0\} \cup Q'_R$
4. let $\Delta = \cup_{(t \to q) \in \Delta_0}(\{add'(t) \to add'(q)\} \cup \{rm'(add'(t)) \to q\}) \cup \Delta_R$
5. return $\mathcal{A} = (\Sigma,\, Q \cup Q',\, Q_f,\, \Delta,\, rm')$

In Step 4, $\{rm'(add'(t)) \to q\}$ is for ensuring μ-compatibility (note that $q = rm'(add'(q))$).

Example 7. Let $R = \{f(x) \to g(x)\}$ and $\mu(f) = \emptyset$, $\mu(h) = \{1\}$. Note that $Q_R = Q'_R = \Delta_R = \emptyset$.
Let $\mathcal{A}_0$ s.t. $Q_0 = \{q_a, q_f\}$, $Q_f = \{q_f\}$, $\Delta_0 = \{a \to q_a,\, f(q_a) \to q_f,\, h(q_a) \to q_f\}$.
The language recognized by $\mathcal{A}_0$ is $L(\mathcal{A}_0) = \{f(a), h(a)\}$.
Then $\mathsf{Init}_{(R,\mu)}(\mathcal{A}_0)$ returns the automaton $\mathcal{A}$ s.t. $Q = \{q_a, q_f\}$, $Q' = \{q'_a, q'_f\}$, $rm'(q'_a) = q_a$, $rm'(q'_f) = q_f$, and $\Delta = \{a \to q'_a,\, a \to q_a, f(q'_a) \to q'_f, f(q'_a) \to q_f,\, h(q'_a) \to q'_f, h(q_a) \to q_f\}$. Note that $L(\mathcal{A}) = \{f(a), h(a)\} = L(\mathcal{A}_0)$.

Lemma 3. *$\mathcal{A}$ is μ-compatible and $L(\mathcal{A}_0) \subseteq L(\mathcal{A})$.*

Lemma 4. $\forall t \in T_\Sigma$, $\forall s \in (Q \cup Q') \backslash (Q_R \cup Q'_R)$, $(t \to^*_\Delta s \implies t \to^*_{\Delta_0} rm'(s))$. *Consequently* $L(\mathcal{A}) \subseteq L(\mathcal{A}_0)$.

3.4 Simplification

Roughly, the simplification step consists in replacing each outermost ground subterm of a given rewrite rule right-hand-side r, by its corresponding state in Q_R or Q'_R. Actually, a simplification step simplifies a critical pair.

function $\mathsf{Simplify}(rm'(\sigma r), q)$

1. let us write $PosGout(r) = \{p_1, \dots, p_n\}$
2. then return $(rm'(\sigma(r)[q'_{r,p_1}]_{p_1} \cdots [q'_{r,p_n}]_{p_n}),\ q)$

Example 8. $R = \{h(x) \to r{=}f(x, g(a))\}$, $\mu(f) = \{1, 2\}$, $\mu(g) = \{1\}$. The initialization gives $\Delta_R = \{a \to q'_{r,2.1},\ a \to q_{r,2.1},\ g(q'_{r,2.1}) \to q'_{r,2},\ g(q_{r,2.1}) \to q_{r,2}\}$. Let $\sigma = (x/q_1)$. So $\mathsf{Simplify}(\sigma(r), q) = \mathsf{Simplify}(f(q_1, g(a)), q)$ returns the pair $(f(q_1, q_{r,2}), q)$, and one has $g(a) \to^*_{\Delta_R} q_{r,2}$. Note that the corresponding transition $f(q_1, q_{r,2}) \to q$ is normalized.

More generally, if r is shallow and let $(t, q) = \mathsf{Simplify}(rm'(\sigma r), q)$, then the transition $t \to q$ is normalized. On the other hand, if $PosGout(r) = \emptyset$, then $\mathsf{Simplify}(rm'(\sigma r), q) = (rm'(\sigma r), q)$.

3.5 Reduction

Le (t, s) be a pair, whose corresponding transition $t \to s$ is not normalized, and suppose that t is reducible into u by non-empty transitions of the automaton. Since the size of u is less than the size of t, it is easier to normalize the pair (u, s) instead of (t, s). The replacement of (t, s) by (u, s) is called *reduction*.
function $\mathsf{Reduce}_{\mathcal{A}}(t, s)$

1. if $t \to s$ is not normalized
   and there exists a non-empty transition $(t_1 \to s_1) \in \Delta$ s.t. $t \to_{[p,\, t_1 \to s_1]} u$
   and $[(p \in Pos^{\mu}(t) \wedge s_1 \in Q) \vee (p \notin Pos^{\mu}(t) \wedge s_1 \in Q')]$
2. then return $\mathsf{Reduce}_{\mathcal{A}}(u, s)$
3. else return (t, s)

In Step 1, if there exist several transitions like $t_1 \to s_1$ that allow to reduce t, then one of them is chosen arbitrarily.

Example 9. $\Delta = \{s(q_1) \to q_2,\ g(q_2, q_3) \to q_4\}$, $\mu(f) = \mu(s) = \{1\}$, $\mu(g) = \{1, 2\}$. Then $\mathsf{Reduce}_{\mathcal{A}}(f(g(s(q_1), q_3)), q) = (f(q_4), q)$.

3.6 Completion

The main algorithm of our method is presented here.

Input: a normalized automaton $\mathcal{A}_0 = (\Sigma, Q_0, Q_f, \Delta_0)$ and a left-linear context-sensitive rewrite system (R, μ).
Output: a context-sensitive automaton $\mathcal{A}$ such that $R^*_{\mu}(L(\mathcal{A}_0)) \subseteq L(\mathcal{A})$.

The main two steps of the algorithm are:

1. $\mathcal{A} = \mathsf{Init}_{(R,\mu)}(\mathcal{A}_0)$. Let us write $\mathcal{A} = (\Sigma, Q \cup Q', Q_f, \Delta, rm')$.
2. while there exists a non-convergent critical pair (cpl, cpr) in $\mathcal{A}$ do
   2.1. $\Delta = \Delta \cup \mathsf{Norm}_{\mathcal{A}}(\mathsf{Reduce}_{\mathcal{A}}(\mathsf{Simplify}(cpl, cpr)))$

Theorem 2. *Let (R, μ) be a left-linear context-sensitive rewrite system, and $\mathcal{A}_0$ be a normalized automaton. When the algorithm stops, $L(\mathcal{A})$ is closed by context-sensitive rewriting and $R^*_{\mu}(L(\mathcal{A}_0)) \subseteq L(\mathcal{A})$.*

Note that it is always possible to make completion terminate, for example by fixing a bound for the number of states. And if this bound is reached, $\mathsf{Norm}_{\mathcal{A}}$ should re-use existing states instead of creating new ones.

However, if the rewrite system is right-shallow, the transition obtained after applying $\mathsf{Simplify}$ is already normalized (see Section 3.4). Then $\mathsf{Reduce}_{\mathcal{A}}$ and $\mathsf{Norm}_{\mathcal{A}}$ do nothing, and no new states are introduced. Therefore, the completion algorithm will stop and generate an automaton similar to that of [14]. Consequently, using the result of [14] we get:

Corollary 1. *If the context-sensitive rewrite system is linear and right-shallow, then the completion algorithm stops and generates the context-sensitive descendants in an exact way.*

4 Examples

The following example shows the role of the states with primes, and of the simplification step.

Example 10. $R = \{h(x) \to r = f(x, h(b))\}$, $\mu(h) = \mu(f) = \mu(s) = \emptyset$.
Let $\mathcal{A}_0$ be the automaton defined by $Q_0 = \{q_a, q_f\}$, $Q_f = \{q_f\}$, and $\Delta_0 = \{a \to q_a,\ s(q_a) \to q_a,\ h(q_a) \to q_f\}$. Note that $L(\mathcal{A}_0) = \{h(s^n(a)) \mid n \in \mathbb{N}\}$ and $R^*_\mu(L(\mathcal{A}_0)) = \{h(s^n(a)),\ f(s^n(a), h(b)) \mid n \in \mathbb{N}\}$ where s^n denotes n occurrences of s.

The initialization step gives:
$Q_R = \{q_{r,2}, q_{r,2.1}\}$, $Q'_R = \{q'_{r,2}, q'_{r,2.1}\}$, $rm'(q'_{r,2.1}) = q_{r,2.1}$, $rm'(q'_{r,2}) = q_{r,2}$,
$\Delta_R = \{b \to q'_{r,2.1},\ b \to q_{r,2.1},\ h(q'_{r,2.1}) \to q'_{r,2},\ h(q'_{r,2.1}) \to q_{r,2}\}$,
$Q' = \{q'_a, q'_f\}$, $rm'(q'_a) = q_a$, $rm'(q'_f) = q_f$,
$\Delta = \{a \to q'_a,\ a \to q_a,\ s(q'_a) \to q'_a,\ s(q'_a) \to q_a,\ h(q'_a) \to q'_f,\ h(q'_a) \to q_f\} \cup \Delta_R$.
With $h(q'_a) \to q_f$ and the rewrite rule, we get the critical pair $(f(q'_a, h(b)), q_f)$. Then $\mathsf{Simplify}(f(q'_a, h(b)), q_f) = (f(q'_a, q'_{r,2}), q_f)$, and the normalization will add the transition $f(q'_a, q'_{r,2}) \to q_f$ to Δ.
$h(q'_{r,2.1}) \to q_{r,2}$ and the rewrite rule generate the critical pair $(f(q'_{r,2.1}, h(b)), q_{r,2})$. Then $\mathsf{Simplify}(f(q'_{r,2.1}, h(b)), q_{r,2}) = (f(q'_{r,2.1}, q'_{r,2}), q_{r,2})$, and the normalization will add the transition $f(q'_{r,2.1}, q'_{r,2}) \to q_{r,2}$ to Δ.

There is no other critical pair. The process stops and the automaton generates $\{h(s^n(a)),\ f(s^n(a), h(b)) \mid n \in \mathbb{N}\} = R^*_\mu(L(\mathcal{A}_0))$. Note that $f(a, f(b, h(b))) \in R^*(L(\mathcal{A}_0))$ whereas $f(a, f(b, h(b))) \notin R^*_\mu(L(\mathcal{A}_0))$, i.e. $R^*_\mu(L(\mathcal{A}_0)) \neq R^*(L(\mathcal{A}_0))$, and notice that the automaton generates only the elements of $R^*_\mu(L(\mathcal{A}_0))$.

In this example, R is right-shallow, and our completion computes an automaton similar to that of [14][5].

The following rewrite system is not right-shallow, and shows the role of the reduction step.

[5] In [14], a tilde is used instead of a prime, but tilde over a state means that a rewrite step is allowed, whereas in our approach a prime means that rewriting is forbidden.

Example 11. $R = \{f(x) \to s(f(x))\}$, $\mu(f) = \mu(s) = \{1\}$. Let $\mathcal{A}_0$ be the automaton defined by $Q_0 = \{q_a, q_f\}$, $Q_f = \{q_f\}$, and $\Delta_0 = \{a \to q_a,\ f(q_a) \to q_f\}$. Note that $L(\mathcal{A}_0) = \{f(a)\}$ and $R^*_\mu(L(\mathcal{A}_0)) = \{s^n(f(a)) \mid n \in \mathbb{N}\}$ where s^n denotes n occurrences of s.

The initialization step gives $Q_R = Q'_R = \Delta_R = \emptyset$, $Q' = \{q'_a, q'_f\}$, $rm'(q'_a) = q_a$, $rm'(q'_f) = q_f$, $\Delta = \{a \to q'_a,\ a \to q_a,\ f(q'_a) \to q'_f,\ f(q_a) \to q_f\}$.
With $f(q_a) \to_\Delta q_f$ and $f(q_a) \to_R s(f(q_a))$, we get the critical pair $(s(f(q_a)), q_f)$. However $s(f(q_a)) \to_\Delta s(q_f)$, i.e. $\mathsf{Reduce}_\mathcal{A}(s(f(q_a)), q_f) = (s(q_f), q_f)$, and the normalized transition $s(q_f) \to q_f$ is added to Δ. No more critical pairs are detected, and the algorithm stops. Now the automaton generates $\{s^n(f(a)) \mid n \in \mathbb{N}\} = R^*_\mu(L(\mathcal{A}_0))$.

If $\mathsf{Reduce}_\mathcal{A}$ were not applied, then the critical pair $(s(f(q_a)), q_f)$ would be normalized into the transitions $s(q_1) \to q_f$, $f(q_a) \to q_1$. But there would be one more critical pair due to $f(q_a) \to q_1$, which would add some more transitions, and so on. In this case, if the normalization process always introduces new states, the completion process would not terminate.

The following rewrite system is not right-shallow, and shows what happens when a subterm forbidden by μ becomes allowed by μ after applying a rewrite step.

Example 12.
Let $R = \{f(x, y) \to h(s(x), s(y))\}$, with $\mu(f) = \emptyset$, $\mu(h) = \{1\}$, $\mu(s) = \{1\}$. Let $\mathcal{A}_0$ defined by $Q_0 = \{q\}$, $Q_f = \{q\}$, and $\Delta_0 = \{a \to q,\ f(q, q) \to q\}$. Thus $L(\mathcal{A}_0) = \{a,\ f(a, a),\ f(f(a, a), a),\ f(a, f(a, a)),\ f(f(a, a), f(a, a)) \ldots\}$. $R^*_\mu(L(\mathcal{A}_0))$ is obtained from the terms of $L(\mathcal{A}_0)$, by replacing some occurrences of f by the pattern $h(s(), s())$ along the left branch, starting from the root. For example $h(s(a), s(a))$, $h(s(f(a, a)), s(a))$, $h(s(h(s(a), s(a))), s(a))$, $h(s(f(a, a)), s(f(a, a)))$ are in $R^*_\mu(L(\mathcal{A}_0))$, whereas $h(s(a), h(s(a), s(a)))$ is not in $R^*_\mu(L(\mathcal{A}_0))$.
The initialization step gives $Q_R = Q'_R = \Delta_R = \emptyset$, and

$$Q' = \{q'\},\ rm'(q') = q, \text{ and}$$
$$\Delta = \{a \to q',\ a \to q,\ f(q', q') \to q',\ f(q', q') \to q\}.$$

Using $f(q', q') \to q$ and $f(x, y) \to h(s(x), s(y))$, we get the critical pair $(rm'(h(s(q'), s(q'))), q) = (h(s(q), s(q')), q)$. This critical pair is not convergent, and cannot be simplified nor reduced. It is not normalized. Then $\mathsf{Norm}_\mathcal{A}$ creates the transitions $h(q_1, q'_1) \to q$, $s(q) \to q_1$, $s(q') \to q'_1$, and add them to Δ.

No more critical pair is detected, then the algorithm stops with $\Delta =$

$$\{a \to q',\ a \to q,\ f(q', q') \to q',\ f(q', q') \to q,\ h(q_1, q'_1) \to q,\ s(q) \to q_1,\ s(q') \to q'_1\}$$

Now, one can see that $L(\mathcal{A}) = R^*_\mu(L(\mathcal{A}_0))$.

In the previous examples $L(\mathcal{A}) = R^*_\mu(L(\mathcal{A}_0))$. However, one may have $L(\mathcal{A}) \supset R^*_\mu(L(\mathcal{A}_0))$, i.e. a strict over-approximation.

Example 13. Let $I = \{f(a, b)\}$ and $R = \{f(x, y) \to f(s(x), p(y)),\ b \to c\}$, with $\mu(f) = \mu(s) = \mu(p) = \emptyset$. Then $R^*_\mu(I) = \{f(s^n(a), p^n(b)) \mid n \in \mathbb{N}\}$ is not a regular tree language and then it cannot be expressed by a tree automaton. So completion will necessarily lead to a strict over-approximation, by losing the link between the number of s and the number of p. Nevertheless, the elements of $\{f(s^n(a), p^n(c))\}$, which are in $R^*(I)$ but not in $R^*_\mu(I)$, will not be generated.

5 Related Work

To the best of our knowledge, the only method to express descendants by regular languages in the framework of context-sensitive rewriting, is that of Sakai et al. [14]. This method returns a tree automaton that recognizes the set of descendants in an exact way (it is not an over-approximation), assuming that the rewrite system is linear and right-shallow. This is why this method cannot deal with Examples 11 and 12, whose rewrite systems are not right-shallow.

Genet et al. compute an over-approximation of the descendants without strategy [9], or according to the innermost strategy [12]. They do not consider context-sensitive rewriting.

Some results of Falke et al. [5, 7] also deal with context-sensitive rewriting, but they do not study reachability problems. They study termination problems and propose a method for proving inductive theorems. The termination problems are based on dependency pairs, and the inductive theorem prover is based on the inference system of Reddy [6].

6 Further work

With our method, and more generally with every method based on the completion of tree automata, the quality of the approximation highly depends on the way the completion is achieved. When normalizing critical pairs, existing states may be used instead of introducing new ones. This helps to make completion terminate. However, the choice of the states to be re-used is crucial for the quality of the approximation. Some heuristics have been developed for ordinary rewriting. Recently, an heuristic using a set of equations E has been presented [11], and an upper-bound for the approximation is given, which allows to estimate the quality of the approximation. We intend to extend these heuristics to context-sensitive completion, so that they could be used within our method.

Another interesting problem to study is: does our method take the map μ into account in a perfect way? In other words, may our method generate descendants that are not context-sensitive descendants? If the re-use of existing states in the normalization process is allowed, we get an over-approximation, and wrong context-sensitive descendants (that are ordinary descendants) may be generated. What about if the re-use of existing states is forbidden?

When considering ordinary rewriting, the set of descendants of a set of terms I is not a regular tree language, even if I is, except if strong restrictions are assumed over the rewrite system. It is the same when considering context-sensitive rewriting. This is why we cannot compute the descendants in an exact way, except for some particular cases. The use of tree languages more expressive than the regular ones, could lead to more precise computations. It has already been studied for ordinary rewriting [13, 1], but not for context-sensitive rewriting.

References

1. Y. Boichut, J. Chabin, and P. Réty. Over-approximating descendants by synchronized tree languages. In *Proceedings of the International Conference RTA*, volume 21 of *LIPIcs*, pages 128–142, 2013.
2. Y. Boichut, T. Genet, T. Jensen, and L. Le Roux. Rewriting Approximations for Fast Prototyping of Static Analyzers. In *Proceedings of the International Conference RTA*, volume 4533 of *LNCS*, pages 48–62. Springer, 2007.
3. V. Cortier, S. Delaune, and P. Lafourcade. A Survey of Algebraic Properties Used in Cryptographic Protocols. *Journal of Computer Security*, 14(1):1–43, 2006.
4. V. D'Silva, D. Kroening, and G. Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions*, 27(7):1165 – 1178, 2008.
5. S. Falke and D. Kapur. Dependancy Pairs for Rewriting with Non-Free Constructors. In *Proceedings of the 17th International Conference RTA*, volume 4098 of *LNCS*. Springer, 2006.
6. S. Falke and D. Kapur. Inductive decidability using implicit induction. In *Proceedings of the International Conference LPAR*, volume 4256 of *LNCS*. Springer, 2006.
7. S. Falke and D. Kapur. Termination of context-sensitive rewriting with built-in numbers and collection data structures. In Santiago Escobar, editor, *18th Workshop on Functional and (Constraint) Logic Programming (WFLP'09)*, 2009.
8. K. Futatsugi, Joseph A. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Proceedings of the International Symposium POPL*, pages 52–66, 1985.
9. T. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In *Proceedings of the International Conference RTA*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.
10. T. Genet. Reachability Analysis of Rewriting for Software Verification. Université de Rennes 1, 2009. Habilitation document, http://www.irisa.fr/celtique/genet/publications.html.
11. T. Genet and V. Rusu. Equational Approximations for Tree Automata Completion. *Journal of Symbolic Computation*, 45(5):574597, 2010.
12. Thomas Genet and Yann Salmon. Reachability analysis of innermost rewriting. In *Proceedings of the International Conference RTA*, volume 36 of *LIPIcs*, pages 177–193, 2015.
13. J. Kochems and C.-H. Luke Ong. Improved Functional Flow and Reachability Analyses Using Indexed Linear Tree Grammars. In *Proceedings of the International Conference RTA*, volume 10 of *LIPIcs*, pages 187–202, 2011.
14. Y. Kojima and M. Sakai. Innermost Reachability and Context Sensitive Reachability Properties Are Decidable for Linear Right-Shallow Term Rewriting Systems. In *Proceedings of the International Conference RTA*, volume 5117 of *LNCS*. Springer, 2008.
15. S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1), January 1998.

Appendix

A Proof of Lemma 1

By induction on the length n of the derivation $t \rightarrow^*_\Delta s$.
- If $n = 0$ then $t = s$, then $rm'(t) = rm'(s)$. Consequently $rm'(t) \rightarrow^*_\Delta rm'(s)$.
- Otherwise $t \rightarrow^*_\Delta t_0 \rightarrow_\Delta s$ (1).
Let $PosQQ'(t_0)$ be the set of the state-positions in t_0. $PosQQ'(t_0)$ can be divided into allowed positions (by μ) and forbidden positions: let $Pos^\mu(t_0) = \{p_1, \ldots, p_k\}$ and let $PosQQ'(t_0) \backslash Pos^\mu(t_0) = \{u_1, \ldots, u_m\}$.
From Derivation (1), for each $i \in \{1, \ldots, m\}$, $t|_{u_i} \rightarrow^*_\Delta t_0|_{u_i}$. On the other hand, for each $i \in \{1, \ldots, k\}$, $t|_{p_i} \rightarrow^*_\Delta t_0|_{p_i}$ by a derivation whose length is strictly less than n. By induction hypothesis, $rm'(t|_{p_i}) \rightarrow^*_\Delta rm'(t_0|_{p_i})$. Consequently
$rm'(t) = t_0[t|_{u_1}]_{u_1} \cdots [t|_{u_m}]_{u_m}[rm'(t|_{p_1})]_{p_1} \cdots [rm'(t|_{p_k})]_{p_k} \rightarrow^*_\Delta$
$t_0[t_0|_{u_1}]_{u_1} \cdots [t_0|_{u_m}]_{u_m}[rm'(t_0|_{p_1})]_{p_1} \cdots [rm'(t_0|_{p_k})]_{p_k} = rm'(t_0)$.
Since $(t_0 \rightarrow s) \in \Delta$ and $\mathcal{A}$ is μ-compatible, then $rm'(t_0) \rightarrow_\Delta rm'(s)$. Therefore $rm'(t) \rightarrow^*_\Delta rm'(s)$.

B Proof of Theorem 1

Let $t \in L(\mathcal{A})$ and consider one rewrite step $t \rightarrow_{[p, l \rightarrow r, \theta]} t'$ allowed by μ, i.e. $p \in Pos^\mu(t)$.
Then $t|_p = \theta l$. Let us write $Var(l) = \{x_1, \ldots, x_n\}$, and for each i, let p_i be the position[6] of x_i in l. Since $t \in L(\mathcal{A})$, there exists $q_f \in Q_f$ such that $t \rightarrow^*_\Delta q_f$. Since $\mathcal{A}$ is normalized, this derivation can be decomposed into:
$t \rightarrow^*_\Delta t[s_1]_{p.p_1} \cdots [s_n]_{p.p_n} \rightarrow^*_\Delta t[s]_p \rightarrow^*_\Delta q_f$ (1) where $s_1, \ldots s_n, s \in Q \cup Q'$.
Then $l[s_1]_{p_1} \cdots [s_n]_{p_n} \rightarrow^*_\Delta s$.
Let σ be the $(Q \cup Q')$-substitution defined by

$$\forall i \in \{1, \ldots, n\},\ \sigma(x_i) = \left| \begin{array}{l} s_i \text{ if } p_i \notin Pos^\mu(l) \\ rm'(s_i) \text{ otherwise} \end{array} \right.$$

From Lemma 1, $\sigma l = rm'(l[s_1]_{p_1} \cdots [s_n]_{p_n}) \rightarrow^*_\Delta rm'(s)$. Moreover $rm'(s) \in Q$. Then $(rm'(\sigma r), rm'(s))$ is a critical pair, and from the assumptions, it is convergent : $rm'(\sigma r) \rightarrow^*_\Delta rm'(s)$.
From Derivation (1), for each i we can extract $\theta x_i \rightarrow^*_\Delta s_i$. From Lemma 1 $rm'(\theta x_i) \rightarrow^*_\Delta rm'(s_i)$. Since $\theta x_i \in T_\Sigma$, $rm'(\theta x_i) = \theta x_i$. Then $\theta x_i \rightarrow^*_\Delta rm'(s_i)$ and recall that $\theta x_i \rightarrow^*_\Delta s_i$. Consequently $\theta r \rightarrow^*_\Delta rm'(\sigma r) \rightarrow^*_\Delta rm'(s)$. Thus $t' = t[\theta r]_p \rightarrow^*_\Delta t[rm'(s)]_p$ (2).

[6] This position is unique since l is linear.

Since $t \in T_\Sigma$, $rm'(t) = t$. Then $t[rm'(s)]_p = rm'(t[s]_p)$ and from (1) and Lemma 1 $rm'(t[s]_p) \rightarrow^*_\Delta rm'(q_f) = q_f$ (3). From (2) and (3) we get $t' \rightarrow^*_\Delta q_f$, i.e. $t' \in L(\mathcal{A})$.
The proof extends to several rewrite steps by trivial induction on the length of the rewrite derivation.

C Proof of Lemma 2

If the transition $t \rightarrow s$ is already normalized, the result is obvious. Otherwise the proof is by induction on the recursive calls.

D Proof of Lemma 3

By construction $\mathcal{A}$ is μ-compatible.
Let $t \in L(\mathcal{A}_0)$. Then $t \in T_\Sigma$ and $t \rightarrow^*_{\Delta_0} q_f$ with $q_f \in Q_f$ (then $q_f \in Q$). From the definition of Δ, we get $add'(t) \rightarrow^*_\Delta add'(q_f)$. Since $\mathcal{A}$ is μ-compatible, and from Lemma 1, $rm'(add'(t)) \rightarrow^*_\Delta rm'(add'(q_f))$, i.e. $t \rightarrow^*_\Delta q_f$. Then $t \in L(\mathcal{A})$.

E Proof of Lemma 4

By induction on the length of the derivation $t \rightarrow^*_\Delta s$. Since $t \in T_\Sigma$ and s is a state, the derivation includes at least one step: $t \rightarrow^*_\Delta t_0 \rightarrow_\Delta s$. Then $(t_0 \rightarrow s) \in \Delta$. Let $s_1, \ldots, s_k$ be the states of t_0 and let $p_1, \ldots, p_k$ be theirs positions in t_0. Consequently $(t_0[rm'(s_1)]_{p_1} \cdots [rm'(s_k)]_{p_k} \rightarrow rm'(s)) \in \Delta_0$ and $s_1, \ldots, s_k \in (Q \cup Q') \backslash (Q_R \cup Q'_R)$, thus

$$t_0[rm'(s_1)]_{p_1} \cdots [rm'(s_k)]_{p_k} \rightarrow_{\Delta_0} rm'(s)$$

On the other hand, for each i, $t|_{p_i} \rightarrow^*_\Delta s_i$. By induction hypothesis $t|_{p_i} \rightarrow^*_{\Delta_0} rm'(s_i)$. Therefore $t \rightarrow^*_{\Delta_0} t_0[rm'(s_1)]_{p_1} \cdots [rm'(s_k)]_{p_k} \rightarrow_{\Delta_0} rm'(s)$.

F Proof of Theorem 2

From Lemmas 3 and 4, Step 1 does not change the language and generates a μ-compatible and normalized automaton. During Step 2, since Function Norm creates normalized transitions and preserves μ-compatibility, the automaton remains normalized and μ-compatible. When the completion algorithm stops[7], every critical pair is convergent. From Theorem 1, the automaton generates a language closed under context-sensitive rewriting, which contains the initial language. Thus $L(\mathcal{A}_0)) \subseteq L(\mathcal{A})$, then $R^*_\mu(L(\mathcal{A}_0)) \subseteq R^*_\mu(L(\mathcal{A})) = L(\mathcal{A})$.

[7] The algorithm may stop because Function Norm makes critical pairs convergent (Lemma 2).